A Self-Scaling and Self-Configuring Benchmark for Web Servers

A Self-Scaling and Self-Configuring Benchmark for Web Servers

Stephen Manley
Network Appliance

Michael Courage
Microsoft Corporation

Margo Seltzer
Harvard University

World Wide Web clients and servers have become some of the most important applications in our computing base today, and as such, we need realistic and meaningful ways of capturing their performance. Current server benchmarks do not capture the wide variation that we see in servers and are not accurate in their characterization of web traffic. In this paper, we present a self-configuring, scalable benchmark, that generates a server benchmark load based on actual server loads. In contrast to other web benchmarks, our benchmark characterizes request latency instead of focusing exclusively on throughput sensitive metrics. We present our new benchmark, hbench:Web, and demonstrate how it accurately captures the load observed by an actual server. We then go on to show how it can be used to assess how continued growth or changes in the workload will affect future performance. Using existing log histories, we show that these predictions are sufficiently realistic to provide insight into tomorrow's web performance.

1 Introduction

Exponential growth has been the norm in the computer industry for the past decade. However, until the advent of the Web, this growth was observed on the supply side: processing power, memory sizes, and disk density. With the Web, we are, for the first time, experiencing exponential growth in demand: the number of users, the number of hosts, the number of bytes transferred, and the number of bytes published. Without a detailed understanding of the workload introduced by these new users and proper tools for evaluating performance, we run the risk of being overwhelmed by increasing demand.

As is to be expected of any nascent application and technology, the tools for Web performance analysis are still in their infancy. We find three main shortcomings in today's web benchmarking:

- The benchmarks do not accurately reflect real usage patterns.
- Today's benchmarks focus on throughput, to the exclusion of response time.
- Today's benchmarks exclude important classes of requests, such as errors and CGI.

Most of the shortcomings in the existing benchmarks stem from the fact that they reflect an understanding of some workload that existed in the past. Due to the extraordinary growth rate that we observe in today's web (we have observed sites whose traffic doubles monthly for extended periods of time), any such workload is obsolete shortly after it is observed, and almost certainly before it has been

standardized, accepted, and used. We have analyzed a large number of web sites and their log histories and concluded that, using a simple set of heuristics, it is possible to create a benchmark that accurately models the traffic that a site currently observes, and more importantly, the traffic that a site will observe, if its growth continues in a similar fashion.

A secondary flaw in much of today's current web benchmarking is the focus on throughput and connections per second. While these numbers can be relevant and interesting, and can help system designers understand the bottlenecks in their system, they do not accurately reflect users' perceptions of performance. As argued by Endo et al. [5], users perceive performance with respect to latency, and presumably, those who set up web servers are doing so to reach users. We have found that when running standard benchmarks, while the results show an impressive number of simultaneous connections, user latencies approach tens of minutes. Such metrics are meaningless, because users rapidly leave a site when they experience such latencies [3]. As such, the impressive numbers reported by the benchmark have little relationship to reality. Thus, in measuring the performance of web servers, we examine the request latencies generated by common web traffic.

Finally, today's benchmarks do not capture the wide variety of requests and responses that servers observe and exhibit. For example, most benchmarks overlook dynamic content, such as that created by the Common Gateway Interface (CGI), which can account for a significant fraction of traffic on sites such as Search Engines. An accurate analysis of CGI poses interesting challenges. First, there is no limit to the number and type of programs that could run as CGI scripts. While CGI can be much more resource intensive for a server (since it typically must spawn a new process), as we are focusing on end-user latency, this effect should only be visible if the increased demands on the server translate into increased latency for the user. Therefore, a good web benchmark should include dynamic content such as CGI, but should model it at a level appropriate to the site being modeled, and evaluate it relative to its impact on end-user latency.

This paper describes hbench:Web, a new web server benchmark that is self-configuring and self-scaling. Without end-user intervention, the benchmark accurately models a server load for which log files are available. The self-scaling nature permits the evaluation of loads that do not currently exist, but which are likely to exist given a particular site history. It also permits the quantitative evaluation of "what if" queries, such as, "What if we make 10% of our site CGI driver?" or "How will end users perceive our site differently if we change servers from Apache to Netscape?"

In the remainder of the paper, we describe our new benchmark and demonstrate how it accurately reproduces the load observed on real servers and how those loads can be modified in a manner consistent with actual usage. In Section 2, we discuss related work on web benchmarking and workload analysis. In Section 3, we describe our benchmarking methodology. In Section 4, we validate our benchmark, and in Section 5 we conclude.

2 Related Work

In order for a Web Server benchmark to be useful, it must produce results that reflect the reality of the Web, and it must do so in a reproducible, statistically significant manner. Unfortunately, the most widely-used benchmarks today fall far short in both categories.

WebStone2.0 [9] is a product created by Silicon Graphics, Incorporated. The first standard in Web server performance benchmarking, WebStone simulates an arbitrary number of clients randomly downloading pages from a server for a parameterized period of time. The key components of the benchmark are a file set, the number of processes downloading files, and the length of the test run. Since it is difficult to define a standard WWW load, these three parameters are user-settable. In theory, users can test server performance under conditions similar to those experienced by their sites. However, the lack of a standardized load hinders the reproducibility and comparability of benchmark results. Each vendor configures the tests differently. Furthermore, there is no simple way to derive a suitable configuration for a particular site. As is often the case, individuals frequently have inaccurate perceptions of the inner workings of their system [7].

WebStone, of course, attempts to define a standard configuration, so the results can be comparable. However, using the standard configuration, WebStone fails to reflect reality in either its file set or its model of user behavior. WebStone measures throughput with only five files on the server; regardless of the user-settable parameters, the average size of the files referenced in the benchmark is much larger than any averages we (or other groups) have observed in either client or server logs. The performance metric reported by WebStone is the bandwidth observed when users constantly download one file at a time, as rapidly as possible. This pattern neither corresponds to any pattern in reality nor does it correspond to the pattern that occurs when multiple users access a site simultaneously. This benchmark ignores the fact that Web sites have different distributions of files and that users rarely behave like WebStone clients.

SPECWeb `96 [8] attempts to rectify the lack of comparability between benchmarking runs, by specifying a canonical WWW load. SPEC's benchmark is based on a small set of configurations. These

configurations allow results from different parties to be compared. In an effort to make the configurations reflect current web usage, the SPECWeb loads are based upon an examination of “current” server logs and are periodically updated. Of course, an accurate definition of “current” changes almost daily and certainly more quickly than standards’ organizations agree on new loads, and the small set of configurations means that SPEC is attempting to reduce all server activity to one of a limited number of general categories. Unfortunately, this is not a simple task; there are a wide variety of servers on the web today; our own log analysis has revealed at least six different categories of web sites and workloads differ by several orders of magnitude within any category. The potential advantage that SPECWeb provides by offering standardized loads becomes an immediate disadvantage if none of the standard loads accurately reflect the load in which a particular individual is interested.

SPECWeb also suffers from metric problems similar to those ascribed to WebStone. SPECWeb’s simulation generates a target number of connections per second, shifting behavior patterns away from those of an actual site. By changing reality, with no understanding of the side effects, the results from the benchmark become less indicative and useful.

Once a benchmark can generate realistic approximations of Web traffic, the statistics it collects become significant. Correct statistics, if focusing on unimportant issues, contribute little to the understanding of Web server performance. While throughput and connections per second provide interesting measures, hbench:Web focuses on the user perception of request handling latency. The logic behind the choice is simple. First, as argued by Endo et al [5], users care primarily about latency. Second, the largest web server vendor claims that user-perceived performance drives development in all their products [6]. From our own web log analysis, 80% of users who experience latency of greater than 15 seconds for a request, immediately leave the site [3]. Web users show a great deal of sensitivity to latency. Since Web site developers aim to attract clients, if their clients cannot use the site effectively, the site has failed. Thus, both from a client and webmaster perspective, the importance of latency exceeds that of all other measurements.

A widely applicable WWW benchmark must be able to produce a variety of different server loads, while still modeling realistic system behavior. The benchmark we describe, hbench:Web, incorporates the strengths of WebStone’s configurability and SPECWeb’s modeling of real WWW traffic. Our benchmark is based loosely on WebStone, but rather than allowing the user to specify the WebStone settings, we create a

representative load based upon analysis of server logs. In essence, we have created an automatic means for creating the representativeness of SpecWeb `96. The key advantages of hbench:Web are first, that its self-configuration makes it possible to measure a server based on a representative load for that server; second, it models all classes of traffic observed on a site (as opposed to being restricted to HTML and image traffic)[1]; and third, it provides the ability to scale traffic to future predicted loads and change the composition of a workload, without damaging the realism of its workload.

A second set of work related to benchmarking is the workload characterization effort. The OCEANS group at Boston University has conducted a number of studies showing that Web traffic is self-similar and that the distributions of file sizes transferred and unique file sizes can be modeled analytically, with a high degree of accuracy [4]. In later work [1], they derived a model for the temporal locality of web accesses. These studies provide valuable insight into understanding workloads, but they do not provide any simple tools or methodology for deriving correct parameters and have not been shown to generalize to a variety of different sites. Given the heterogeneous nature of the Web, it is not obvious that any single distribution model will be generally applicable to a wide range of sites.

3 Deriving Workload Models from Server Logs

In earlier work [2], we conducted extensive analysis of a wide range of server logs to derive a taxonomy for web sites and understand how web sites grow and differ. Due to the agreements with some of our log providers, we are unable to reveal the specific sites for which the logs were collected, but we can describe them in terms of their important characteristics. These summaries are in Table 1. Unsurprisingly,

Anonymized Site Name	Content	Server Software	Time	Requests/ Month	MB on Site
Traditional Business	Information on subject matter, advertisements	Apache 1.1.3	1/96 - 2/97	321,747	2.8
University Departmental Server 1	Graduate student Web pages, department information	NCSA 1.4.2	4/96-2/97	106,001	196.0
Central University Server	Information for an academic institution, student Web pages	NCSA 1.4.2 Apache 1.1.3	10/94-2/96	2,328,401	455.0
ISP company page	Simple advertisement	Apache 1.1.3	9/96 - 2/97	8,139	1.5
University Departmental Server 2	Graduate student Web pages, department information	Netscape Netsite-Commerce 1.0	7/95-12/96	85,763	115.0

Table 1: Site Survey Description. These logs come from two different universities and two different ISPs.

1. WebStone has recently proposed a CGI benchmarking standard, and SPECWeb ‘98 is rumored to include CGI as well, but neither has been accepted nor made available to date.

Anonymized Site Name	Content	Server Software	Time	Requests/ Month	MB on Site
Adult-Entertainment 1	Adult images, movies, chat-rooms	Apache 1.1.3	3/96-9/96	69,906	135.0
Adult-Entertainment 2	Adult images.	Apache 1.2b8	6/96-5/97	1,264,045	17.9
Organization for Members of same Profession	Articles and images pertaining to field	Microsoft IIS/3.0	4/96-2/97	42,301	0.8
Web site designer	Samples of different sites, games	Apache 1.1.3	8/96-2/97	43,523	0.7
Government Agency	Information on agency's actions	Apache 1.1.3	8/96-2/97	26,049	1.2
Free Web Software Site	Evaluation copy of proprietary Web software	Apache 1.1.3	4/96-1/97	15,982,085	136.0

Table 1: Site Survey Description. These logs come from two different universities and two different ISPs.

we found that sites and their traffic vary tremendously. As a result, a realistic benchmark must be able to model a wide diversity of traffic patterns. Furthermore, as sites change rapidly, a benchmark that creates static models of traffic patterns will soon become outdated. Therefore, a benchmark should be able to generate new models of traffic and constantly update those models. Finally, the benchmark must create simple and redistributable models of traffic. Since one of the values of a benchmark comes from being able to compare different configurations of hardware and software with the same test, the benchmark must be simple.

Under the complexity of the technology and rapid changes on the Web lies a simple model: that of clients sending queries to servers. Clients include proxy servers, search engine robots, and individuals reaching a server via WAN, LAN, or modem. Regardless of the category of client, or the identity of individuals who are clients, each follows the same model: sending requests or posting data to the server. At the same time, Web sites follow a similar model: hosting a set of static and/or dynamically generated documents. These are the files and scripts that are accessed by users, at various intervals. As this model is simple, accurate, and constant, it is the basis of hbench:Web.

Consistent with this basis, we have implemented a benchmark that pre-processes actual Web server logs to generate a configuration that, when run through the benchmark, enables the benchmark to model the traffic described by the log. Using server logs to generate traffic models, we can describe the model by three parameters: the site's *page set*, a collection of *user profiles*, and the *inter-arrival interval* between users.

A page set is not the same as the set of files accessed on a server. By pages, we mean logical Web documents, each of which may be comprised of many different files. Our first step in deriving a workload

characterization from a log is to group file requests into logical pages. We use the fact that a user's browser requests the files that comprise a Web page as rapidly as possible; the only latency between requests for files on the same page is that experienced by users in receiving files. In analyzing the server logs, we identify the server latency that captures 80% of the file requests. We then use twice this 80%-latency to establish the upper bound on the delay between two files in the same page. For the logs we analyzed, this yielded a two second latency; our page set detector groups together files that one user accesses within a two second interval.

Our next step is to capture user session profiles. We define a user by an IP address. While this means that a proxy server or multiple clients on one time-sharing machine are considered to be a single user, this aggregation does not detract from our model's realism, since we model the aggregated user set. A user profile captures all of the important information about a user: the requests made to the server, the idle time between each set of requests, and the length of a session. We define a session to be the collection of Web pages, accessed, and the inter-access time between these pages. We consider a session ended if the user remains idle for more than 5 minutes, since our log analysis revealed that if a user is idle for more than 5 minutes, he or she is likely to remain idle for at least a day. In other words, if a user is idle for more than five minutes, a surfing session has been completed.

The third parameter, user inter-arrival time, controls the benchmark load. The number of users sending requests to a server in a given time period has a direct impact on the load experienced by a Web server. This element also conforms to standard Web behavior. Users arrive at a site, download information, go idle temporarily, and then download more information. Eventually, that user leaves the site. While this user is active, however, other users send requests to the server. To effectively select the user inter-arrival time, we divide the log into n/m sublogs where n is the number of seconds of traffic represented by the logs, and m is the length of log time for which the benchmark will run (in our case one hour or 3600 seconds). For each of these sublogs, we determine the average inter-arrival time. This data forms a cumulative distribution so that load can be specified as a percentile of all perceived loads. For example, if the desired configuration is maximum load, the user would specify the 99th percentile hour and simulated users would arrive as rapidly as they arrived in the heaviest hour recorded in the log.

3.1 Statistical Basis for Model

The model, while simple and intuitive, needs mathematical justification. At any given point in time, the traffic on a site can be described by the users and the files they access. This information is captured by the log. However, rather than selecting parts of the log to re-run, one would prefer to be able to create an abstraction of the log's traffic and be able to select different levels of load, without having to search the log for a segment that generates the desired load. This permits the benchmark to generate loads that were never actually observed, but which share important statistical properties with actual loads. The structure of the model forces the set of files and users to remain constant, while the rate at which users access the site changes. Therefore, the model can only be considered valid if each subset of traffic in a log exhibits similarity to the other traffic patterns and the log taken as a whole.

In determining the similarity of different segments of the traffic, we need to determine which aspects of traffic are most important to track. In this case, the focus should be on the type of traffic, since the amount of traffic is controlled by a non-static aspect of the benchmark. Five factors, other than load, characterize a site's traffic pattern. The similarity in the page set can be seen in the distribution of file sizes and file types by request. The similarity in user set can be determined by the distribution of number of requests per user. The similarity in general traffic patterns can be seen in the percentage of requests per method (GET, HEAD, POST), as well as percentage of requests per server response (e.g., "200-OK", "304-Not Modified", "404-Not Found"). From these factors, we can determine whether the traffic on a site follows set patterns, regardless of the number of users accessing the site.

A chi-square test of homogeneity quantifies the degree of similarity between each of the factors of traffic similarity. The chi-square test of homogeneity can compute the goodness-of-fit between two multinomial distributions. A set of discrete buckets forms the multinomial distribution for this test. Two of the factors (method type, response type) already fit into buckets. Most file types are rarely accessed, so file types were divided into buckets of: HTML, CGI, images, and other. The file sizes were quantized into buckets of size less than: 500 bytes, 1KB, 5KB, 10KB, 50KB, 100KB, 500 KB, and infinity. The granularity of bucket sizes decreases as file size increases. This reflects the observed pattern of many small files and a few large files in the server logs. The number of requests by user are quantized into buckets of size less than: 5 requests, 10 requests, every ten requests up until 100, and greater than 100 requests. The

chi-square test then compares relative frequency of elements in buckets to create a metric for the similarity of distributions.

In order to show that these parameters do describe the traffic patterns observed in logs, we extracted ten sublogs from each log by selecting a random point in the log and accumulating an hour's worth of requests. Each sublog consists of approximately 0.5% of the site's total requests. The traffic was then quantized, and the chi-square test conducted. The chi-square test convincingly shows that a site's traffic patterns over a period of a week show little variance. For each site, the chi-square test shows over 99% similarity between ten sub logs and the overall log. In each case, on each log, the chi-square test indicated that the similarity between sites was greater than 99%. Essentially, the test declares that the model of each sublog is the same because the variance observed in the different sublogs can be attributed to random variation over 99% of the time. This data agrees with Crovella and Bestavros' results about self-similarity [4] and indicates that the model of the Web site does remain valid for any given time period to be examined.

To further prove the model's accuracy, there must be a relationship between idle time between users and server load. The analysis of the constancy of factors such as number of requests per user, and the distribution of file requests by size indicates that the only way to adjust load would be to add users. As a verification, for each month of the logs, the log was broken into hour-long sublogs. In each month, the hour with the greatest number of requests always ran the greatest number of users in that hour. The same did not hold for bytes transferred. In some of the lightly loaded hours (the lowest 50% in the log), a few large requests led to more traffic than many small requests. However, the pattern held in all cases with significant traffic. The statistical analysis shows that the model of the monthly log corresponds closely to the smaller increments of traffic, and that the number of users does determine the level of traffic.

3.2 Reducing the Model

On a large site, the model can become extremely large. The set of pages can reach the thousands and consume hundreds of megabytes of space. For a benchmark, the act of reconstructing an entire file set can be daunting. Furthermore, the space devoted to retaining user logs can become nearly as large as the original log - which can be as many as eighty megabytes. Such a model can be infeasible to retain, share, or even run as a benchmark. Therefore, the model must be reduced. Fortunately, we can do by means of some simple heuristics.

3.2.1 File Set Reduction

We use two levels of file set reduction. Both levels use template model pages and place all files using the same template in an equivalence class. The template specifies two parameters, the percentage of identical files and the acceptable variation in size of unlike files. For example, for the first level reduction, we used a template requiring 75% of the files in a page to be identical and the remaining 25% of the files to be within a factor of 2 in size of their counterparts (that is, for each unlike file, it must be between half as large and twice as large as its counterpart in another document). The 75% threshold was derived by examining a several log files. The factor of two in size came from further log examination, focusing attention on the size of in-lined GIF images. In both cases, we derived the parameters by analyzing a few "training" logs and then running experiments on other "test" logs. Once we have performed the file reduction, we retain one file in each equivalence class, rather than retaining all files in the file set. This causes little change in the actual file set, except for possible side effects in changing the percentage of types of files, such as HTML, CGI, or image files.

The second level of reduction leads to a great departure from the original model. We use the template parameterized such that 25% of the files are identical and the remaining 75% are within a factor of two in size. Table 2 shows the reduction we obtained for each of our sample traces using the two different reduction levels.

	% Reduction (MB Reduction)		
	Free Software Site	**Adult Entertainment 2**	**Central University Server**
Level 1	20% (27.0)	55%	9% (41.0)
Level 2	80% (86.4)	87%	47% (194.6)

Table 2: File Set Reduction. Note that the level 2 results are cumulative in that they show a reduction after the level 1 reduction has already been applied.

3.2.2 User Set Reduction

Similar to the file set reduction, we use two levels of user profile reduction as well. As we created equivalence classes for files, we create equivalence classes for users, weighting each class by the number of users that comprise it. In creating the user set, users with identical access patterns (i.e., those users who access the same pages in the same order) are always combined. The first reduction level combines all users

who access the same pages, but in different order. The second reduction level combines all users who request similar pages. Pages are defined as similar when they have between 50% and 200% of the same number of files as another Web page, and the sizes of the files that comprise these pages are also between 50% and 200% of one another. As in the case of file set reduction, the second level of reduction significantly alters the original model. Once again, we used server log analysis to help us select parameters. We observed a few different types of user behavior: shallow surfing, deep exploration, search engine traversal, etc. Table 3 shows the reduction we obtained for each of our sample traces using the two different reduction levels.

	% Reduction (Reduction in Number of Profiles)		
	Free Software Site	**Adult Entertainment 2**	**Central University Server**
Level 1	11% (203)	3% (36)	7% (162)
Level 2	40% (656)	86% (1004)	80% (1718)

Table 3: User Set Reduction. Note that the level 2 results are cumulative in that they show a reduction after the level 1 reduction has already been applied.

3.3 Changing Traffic Patterns

While a realistic benchmark helps provide good analyses of current server performance, researchers, companies, and webmasters often want to test the effects of a new development. The simple self-configuring model discussed so far allows the user to set the load generated as a fraction of the original load on the server. However, it does not provide the ability to scale beyond the original load or to change the original workload (e.g., what would happen if we made all our pages dynamically generated?). To augment our basic benchmark, we allow a user to specify the ratio of traffic for different types of requests. Our goal is to provide users the ability to change some aspects of traffic patterns while retaining as much of the real traffic pattern as possible.

Category	**Traffic Type to Specify**
Method of Accessing Server	GET, POST, HEAD, DELETE, TRACE
Server Response	200 (OK), 302 (Redirect), 304 (Not Modified), 403 (Not Permitted), 404 (Not Available)

Table 4: Available means of altering traffic patterns. Note that each of these types of traffic is changed, not at a file level, but by selecting user profiles with the desired characteristics

Category	Traffic Type to Specify
File Type	HTML, CGI, Image, Other
File Size	Less than 500 bytes, 1KB, 5KB, 10KB, 50KB, 100Kb, Larger

Table 4: Available means of altering traffic patterns. Note that each of these types of traffic is changed, not at a file level, but by selecting user profiles with the desired characteristics

Traffic modification could be accomplished in two ways: altering the file set or altering the user profiles. Table 4 lists the possible traffic modifications. All of the available methods of altering traffic patterns involve changing the type or size of responses to the user. In an ideal world, we could determine exactly the appropriate user behavior to induce such a change. We find that we are able to retain the characteristics of our original trace more successfully by altering the ratio of user profiles. Each user profile models what an actual user does. The profiles, when combined, form the entirety of a site's traffic. Therefore, to change the traffic model, we identify the user profiles that will cause the desired shift in overall traffic. Since the model includes only actual user profiles, this requires changing only the ratio of use of the different user profiles. In other words, the benchmark models the modified traffic pattern as being caused by more clients who behave in a given way.

While this method retains much of the original model, there are flaws in the chosen method. First, if the desired level of traffic is not demonstrated by any user, the model cannot be modified to meet user specifications. In other words, if a researcher wanted to increase a site's CGI traffic from 2% to 50%, and no user profile includes over 50% of requests for CGI, the request will fail. Furthermore, if a limited number of user profiles fits the requirement for the alteration to succeed, creating the altered model could lead to a negative side effect. The reproduced user profiles make requests other than those that are specifically desired. The small number of profiles, and their other requests, could dominate the model of the log, and change non-desired types of behavior. In general, these types of shifts improve the reality of modification—a change in one type of behavior is likely to induce other changes. However, as the set of user profiles used to change the site's traffic decreases, the realism of the new model must be questioned. Despite the flaws, this method of changing traffic patterns fits within the framework of the model. It still retains the notion of users accessing files, and does not presume to generate an "ideal" user, or modify one user's behavior. Instead, it uses actual client profiles to provide a template for modeling the chosen traffic

pattern. This method preserves the variety of side effects that come with changing a site's traffic and retains the other user profiles to preserve the variety of traffic on a site.

3.4 Scaling the Model to Predict Future Traffic Patterns

A second shortcoming mentioned above is that the basic benchmark cannot generate a load greater than that observed in the original logs. However, as the web continues to grow at an exponential rate, it is useful to be able to predict and characterize loads that are likely to occur. We scale workloads in hbench:Web by using historical server logs to project the future traffic pattern for a site. The benchmark requires data from at least three months of a site's traffic, and generates a model of the site's traffic for as many as six months into the future. The model of the benchmark remains unchanged—the future model must also be based on file set, user set, and user arrival times. Of course, any model would struggle to determine changes in a site's set of files, and the behavior of the users. After all, each of these factors are controlled by individuals, who are far too complex to model. On the other hand, user arrival time can be based on a set of millions of Web users. One person's choices become insignificant in the face of such a large set of users. Therefore, scaling focuses on predicting the rate at which new users will access the site.

In Section 4.4, we show that the model of a Web site's traffic remains valid over a six month period. Since the manner in which users accessing files does not fundamentally change, the basic features of the model do not vary, the file set and user profiles will still model real traffic patterns. The user profiles may not be for users in the current month and the file set may be outdated, but the traffic patterns they represent are still valid. Therefore, to model the load on the site, one needs only to modify the inter-arrival time of users. To generate the future arrival times, the benchmark extrapolates on the changes in past arrival times of users, with respect to the growth on the site. In determining the projected change of user arrival time, we compute the percentage change of that factor from one month's model to the next. The simplest solution would then be to average the growth patterns and simulate linear growth, but with the exponential growth and change on the Web, such a linear change would be incorrect. Instead, the weight given to past growth rates decreases exponentially by the rate of growth of traffic on the site.

The strength of the original model enables the benchmark to scale to predict future traffic. Over time, the type of traffic on a site undergoes very little change. While the site grows, the type of content varies little. Still, even such a model can fail if external events significantly alter the site, as happened with the

organizational site. Such a change emphasizes the assets of this model. The model has been built upon the most static elements of a Web site, to minimize the susceptibility to the rapid changes of the Web.

3.5 Run Time Environment

The final challenge for our benchmark is creating realistic traffic and generating meaningful statistics. At the most general level, our benchmark builds upon the WebStone master-client relationship. Unlike WebStone, and the benchmarks that followed, hbench:Web adds a third element—the statistics daemon. Whenever a new user should be generated, the master selects a user profile according to the appropriate distribution, spawns a client process to carry out that user profile, and then sends its data to the statistics daemon.

The use of a separate statistics daemon and post-processing scripts solve many of the problems that plague existing benchmarks (e.g., the master process is often wasting cycles processing statistics rather than generating new clients). Clients connect to the daemon only when they have completed a user profile. The statistics daemon can run on any machine because the port and machine on which the daemon runs has been set up before the benchmark runs. Since statistics are gathered outside the master, multiple master processes can be run on different machines; this increases the power of the benchmark enormously.

Clients send information about the overall session, each page retrieval, and each file request. Reported numbers include: header size, number of packets sent, number of packets received, connection latency, header latency, body latency, page latency (the time for all files to download), number of connections per page (for persistent connection analysis), and most importantly, the time at which each session, page retrieval, and file request began. This data enables the post processor to make a thorough analysis of the issues that affect one user's request. All data is time-stamped permitting a post-mortem analysis of interaction between different user profiles. Various data that can be extracted include: file and page latency by the number of connections, each file request's latency broken into the time spent making the connection, sending the request, receiving the header, and receiving the entirety of the body. Receiving such large amounts of data could potentially make the statistics daemon a bottleneck, however we can run multiple daemons as well, partitioning the client population among them. After the benchmark is complete the multiple daemon output files can be processed to generate final results.

4 Validating the Benchmark

Validating the benchmark equates with verifying that the self-configuration generates a load representative of the original server. We evaluate the benchmark load in three ways. First, we compare the

load generated during the benchmark to the original traffic, using the chi-square tests we used in Section 3.1. Next, we evaluate our ability to modify the file type distribution. Finally, we evaluate the reality of our scaling by using old logs to predict current access patterns.

4.1 Experimental Setup

We evaluate the accuracy of the benchmark with respect to our three largest loads: the Free Software Site, Adult Entertainment Site 2, and Central University Server. We limit the length of the benchmark to 15 minutes with a 5 minute start-up interval. To test the benchmark's ability to accurately induce a requested load, we ran different load levels each time. The projected goals were: the 50th, 60th, 70th, 80th, 90th, and 99th percentile heaviest loaded 15 minute segments from the original month-long log. The data, unless mentioned, varied little. Each of the reported numbers is the most pessimistic observed.

The tests were run on off-the-shelf systems. Our server consists of an Intel Endeavor Motherboard (i430FX chip set) with a 120 MHz Pentium Processor, PCI bus, 8 KB separate instruction and data caches, a 512 KB pipeline burst off-chip cache, and 32 MB of 60 ns EDO memory. Our software consisted of BSD/OS 2.1 and Apache 1.1.3. The web site was hosted on a 1033MB, 5400 RPM Fujitsu M2694ES disk, with a 10ms average seek time and a transfer rate of 5MB/second accessed through a BusLogic BT946C PCI SCSI controller. The Ethernet adapter was a 10Mbps Western Digital Elite 16 with 16KB of 100ns RAM.

Our request generator was a SparcStation 20 with two 75 MHz SuperSparc processors and 128 MB of RAM, running Solaris 2.4. The network subsystem was the built-in 10MB Lance Ethernet. The tests were run on the same local subnetwork, but not a private network.

4.2 Analysis of the Basic Model

Our goal in this section is to show that the traffic generated by the benchmark matched that of the original log. Once again, we resort to the chi-square test described in Section 3.1. As we cannot distinguish between different users during the benchmark run (because they all come from the same machine/IP address), we omitted the number of files per session analysis. However, we found that for all three sites and the four distributions (file type, file size, request types, and response types), the correlation was excellent: usually 99% and always above 95%.

In addition to the chi-square test, we analyzed how effectively we were able to produce the targeted load. We compared the load generated during the benchmark to that of an appropriate interval in the

original load. A fifteen minute test generated enough traffic to create a close approximation to the original traffic. There were only a few areas in which traffic differed. For the Free Software site, we observed 5% more requests for small files (those smaller than 500 bytes) and a corresponding increase in files in the 5KB bucket. On the University site, we observed a 3% increase in HTML documents and a decrease of 2% in images and 1% in CGI. The Adult Entertainment site exhibited the greatest discrepancies with 7% too many 5 KB files and 4% too few 500 bytes files and 3% fewer 50 KB files. These minor discrepancies result from extracting an hour's worth of log data to determine the parameters for a 15-minute benchmarking run. However, the accuracy of the results, combined with the convenience of running the benchmark for only fifteen minutes, indicates that the modeling has been successful.

4.3 Analysis of Generating Specific Types of Traffic

In order to provide users with a planning tool, it is imperative that the benchmark model be flexible enough to allow users to pose queries of the form, "How will my site perform if I increase the amount of CGI traffic by 50%." To test the benchmark's ability in this respect, we retargeted the Free Software Site's model to induce both small and large increases and decreases in certain types of traffic. The results, shown in Table 5, are remarkably positive. We were successful at inducing both large and small increases and decreases in traffic, and achieving ratios nearly identical to our targets. This ability to plan for future site changes can ease the burden of long range planning for web masters.

Traffic Specification	Current Traffic Level	Target Traffic Level	Adjusted Traffic Level
Increase level of "Not modified" Responses - 304	20%	60%	59%
Increase level of "Not modified" Responses - 304	20%	25%	25%
Increase level of CGI traffic	10%	54%	57%
Increase level of "OK" Responses -200	77%	85%	86%
Decrease level of CGI traffic	10%	3%	2%
Decrease level of GET requests	97%	40%	40%

Table 5: Results of Changing Traffic Type Distributions. By altering the ratio of user profiles, we induced specific increases and decreases in traffic types. Such flexibility allows a user to answer queries such as, "What if CGI traffic increases by f0% over the next few months?"

4.4 Analysis of Predicting Future Patterns

Our ability to model future workloads relies on a basic assumption that while sites and user behavior change, fundamental traffic patterns do not undergo major transformations. To test the behavior of the model over time, we conducted a chi-square, goodness of fit test on the most recent 6 months of each server log. The test mirrors that conducted on the original model, except that rather than determining the similarity of many sets of traffic from one month, it determines the similarity of each month over one-half year. The results, shown in Table 6, indicate that the fundamental features of file set and user behavior remain constant. In fact, only the organizational site has changed significantly. As it turns out, the webmaster of the organizational site changed the entire model of the site—charging for access and driving the site through a CGI interface. Still, even sites such as the traditional business, which exhibited two tremendous growth spurts due to a site overhaul, do not show fundamental changes in traffic patterns.

To analyze the scaling behavior of our benchmark, we used the first three months of logs to project one, three, and six months into the future. The resulting traffic is remarkably similar to that of the original log. We focus on two aspects of the scaling: the effectiveness of scaling user inter-arrival times and how closely the resulting traffic approximates reality.

Site Name	Correlation in Dist.of File sizes	Correlation in File Type	Correlation in Dist. of Files per User	Correlation in Server Responses	Correlation in Client Methods
Trad. Business	99%	99%	95%	99%	99%
University Departmental Server 1	95%	97.5%	94%	96%	97.5%
Central University Server	98%	99%	99%	96%	99%
ISP	99%	99%	99%	99%	99%
University Departmental Server 2	85%	99%	97%	99%	95%
Adult Entertainment 1	**96%**	**5%**	**18%**	**96%**	**90%**
Adult Entertainment 2	95%	99%	98%	99%	99%
Adult Entertainment 3	97%	95%	93%	98%	99%
Organization	**5%**	**10%**	**21%**	**87%**	**5%**
Web Site Designer	99%	99%	96%	99%	99%
Gov’t Agency	99%	99%	93%	99%	99%
Free Software	97%	99%	99%	97%	98%

Table 6: The result of a chi-square test for similarity of six months of sites. For virtually every site, the model of traffic scarcely changes over the six month period. Only the Organization and Adult Entertainment 1 sites show poor correlation. The Organization site underwent a dramatic transformation, and the Adult Entertainment 1 site was on its way to extinction during the analysis period.

4.4.1 Scaling Inter-arrival Times

Using the three month history of inter-arrival times, we scale the future inter-arrival times to model consistent growth. This method of scaling achieved moderate success. Table 7 shows the projected inter-arrival times compared to the inter-arrival times observed in the logs.

Months into the future	Inter-Arrival Time (in seconds)					
	Free Software Site		University		Adult Entertainment 2	
	Real	Predicted	Real	Predicted	Real	Predicted
One	4.91	4.87	10.1	10.3	4.1	3.9
Three	2.03	2.41	8.9	9.4	4.4	4.5
Six	1.65	1.84	7.6	8.4	5.0	5.0

Table 7: Predicted and Actual Inter-arrival times. The Adult Entertainment site shows the best prediction, because its users have a consistent pattern. We predict a more lightly loaded University site because our predictions are based on summer traffic. The Free Software Site exhibited an unusual usage spike in month three, which results in less accurate predictions.

For the Free Software Site, the three-month arrival time shows a slight deviation that has not been fully recovered in the sixth month. While the three month time is off by nearly 20%, the six month time is off by only 10%. Further analysis of the logs show that the site underwent a user spike in the third month that could not be predicted, however the partial recovery by the sixth month is rather impressive. Given the value of projecting several months into the future, we find the projected inter-arrival times satisfactory.

The University site predictions are also somewhat off, and this is due to poor selection of “training” months. The three months analyzed were June, July, and August, while those predicted fell during the academic year. This results in our under-predicting the growth rate, resulting in somewhat higher inter-arrival times than were observed.

Our ability to predict the future load on the adult entertainment site is not entirely surprising. The pattern on this site is extremely regular and predictable; most users stop and view the “free pages” and leave the site once they are required to become members and pay for viewing more of the site. This pattern changes very little; although total traffic decreases throughout our measurement period, the users remaining follow the same access patterns followed previously.

Overall, our ability to predict future loads is a function of the regularity of a site's pattern, but even in the face of adverse conditions (e.g., an unexpected user spike in month 3 or "training" on an inappropriate set), our model comes acceptably close to predicting actual traffic loads.

4.4.2 Traffic Similarity

Our last test once again uses the chi-square test to evaluate how closely our benchmark traffic represents the traffic actually depicted in the logs. The traffic generated by the benchmark does not mirror reality exactly, but achieves great success in most cases. Table 8 presents the results of the standard chi-square tests on all three sites for the one, three, and six month intervals. As expected, the benchmark's accuracy deteriorates over time, but it never falls below 80% similarity to the original model in any category of traffic. The same explanations that appear in Section 4.4.1. apply here as well. The three months of logs for the University site were inappropriate for projecting the months we did. The Free Software Site exhibited an unexpected growth spurt in month 3 that we did not capture, but the constancy of the Adult Entertainment site was captured beautifully.

	Number of months into the future	Correlation in Bytes per Load	Correlation in Requests per Load	Correlation in File Type	Correlation in Dist. of File Size
Free Software Site	Month 1	94%	96%	98%	98%
	Month 3	86%	87%	97%	98%
	Month 6	85%	84%	90%	93%
University Site	Month 1	90%	97%	97%	88%
	Month 3	87%	89%	98%	85%
	Month 6	85%	88%	97%	81%
Adult Entertainment Site	Month 1	98%	98%	99%	98%
	Month 3	96%	96%	99%	97%
	Month 6	92%	93%	98%	94%

Table 8: The scaled model's goodness-of-fit with the actual traffic patterns. The model is remarkably effective, creating very accurate simulations of future traffic.

5 Conclusions

We have demonstrated a benchmark capable of accurately modeling existing Web traffic, predicting future workloads, and providing users the ability to evaluate changes in their workload. The benchmark is

self-configuring and self-scaling, removing the guesswork from setting parameters found in other benchmarks or models.

This benchmark, in conjunction with a representative Web taxonomy, has the potential to revolutionize the process of Web evaluation. The future of such a benchmark depends on an agreement as to the types of loads that together represent the diversity of sites populating the Web. Such a collection of loads, expressed either as a set of reference logs or a set of reference configurations, will be longer lasting than today's standard workloads, because they can easily be scaled to reflect the overall growth in the Web. While no single load is representative of "The Web," we believe that a collection of loads from the variety of sites such as corporate servers, search engine sites, academic institutions, special interest sites, massively popular sites (e.g., netscape.com), and adult entertainment sites provide the basis for impartial, reproducible comparisons.

6 Availability

Our benchmarking tools, including the configuration generator and benchmarking program are all publicly available. We are in the process of creating configuration scripts for a variety of sites representing the commercial sector, the academic sector, search engine sites, adult entertainment sites, and any other sites that demonstrate access patterns different from those mentioned. These configuration scripts will also be publicly available.

7 Bibliography

[1] Virgilio Almeida, Azer Bestavros, Mark Crovella, Adriana de Oliveira. "Characterizing Reference Locality in the WWW", *Proceedings of IEEE PDIS'96: The International Conference in Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1996.

[2] Manley, S., Seltzer, M., "Web Facts and Fantasy," Proceedings of the USENIX Symposium on Internet Technology and Systems, Monterey, CA, 1997.

[3] Manley, S., "An Analysis of Issues Facing World Wide Web Servers," Harvard University Tech Report, TR-97-12, May 1997.

[4] Mark Crovella, Azer Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *Proceedings of SIGMETRICS 96*, Philadelphia, PA, May 1996.

[5] Y. Endo, Z. Wang, J. Chen, M. Seltzer. "Using Latency to Evaluate Interactive System Performance," *Proceedings of the 1996 Symposium on Operating System Design and Implementation (OSDI)*.

[6] Freier, A., Personal Communication, October 1997.

[7] Saltzer, Jerome H., and Gintell, John W. “The Instrumentation of Multics,” *Communication of the ACM, 13,* No. 8, August 1970, pp. 495–500.

[8] SPECweb96 Benchmark Information can be found at http://open.specbench.org/osg/web96/

[9] G. Trent, M. Sate. WebSTONE: The First Generation in HTTP Server Benchmarking. White paper available at http://www.sgi.com/Products/Web- FORCE/WebStone/paper.html